Loop Level Parallelism and Owner-Compute Rule on Mome, a Relaxed Consistency DSM

Yvon Jégou

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Loop Level Parallelism and Owner-Compute Rule on Mome, a Relaxed Consistency DSM

Yvon JEGOU

N°4058

Décembre 2000

THÈME 1

ISSN 0249-6399 ISRN INRIA/RR--4058--FR+ENG

Loop Level Parallelism and Owner-Compute Rule on Mome, a Relaxed Consistency DSM

Yvon JEGOU

Thème 1 — Réseaux et systèmes
Projet Paris

Rapport de recherche n˚4058 — Décembre 2000 — 20 pages

Abstract: In this paper, we consider the application of the owner-compute rule for the parallelization of HPF-style numerical programs and the use of Mome, a software DSM, as a base for the run-time system. Mome is a page-based relaxed consistency DSM which allows the parallel processors to make consistency requests on individual pages, for instance after synchronization operations. We show that, from the same informations used for explicit message passing run-time systems, it is possible to control the behavior of our DSM and to obtain good performance.

Key-words: DSM, relaxed consistency, HPF, owner-compute rule, data prefetch.

(Résumé : tsvp)

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France)
Téléphone : 02 99 84 71 00 - International : +33 2 99 84 71 00
Télécopie : 02 99 84 71 71 - International : +33 2 99 84 71 71

Exploitation du parallélisme de boucle et de la règle des écritures locales sur Mome, une mémoire virtuelle partagée à consistance relachée

Résumé : Dans ce document, nous considérons l'application de la règle des écritures locales pour l'exécution parallèle de codes numériques de type HPF et l'utilisation de Mome, une mémoire virtuelle partagée (DSM), comme support pour l'exécutif du langage de programmation. Mome est une DSM à consistance relachée qui laisse aux processeurs la charge d'émettre des requêtes de consistance sur les pages, par exemple, après les opérations de synchronisation. Nous montrons qu'en exploitant les mêmes informations que celles utilisées par des exécutifs à échange explicite de messages, il est possible de contrôler la consistance de la mémoire de notre DSM et d'obtenir de bonnes performances.

Mots-clé : mémoire virtuelle partagée, consistance relachée, HPF, règle des écritures locales, préchargement

1 Introduction

The classical compilation scheme for High Performance Fortran (HPF, [7]) is based on the application of the owner-compute rule which, after distributing the ownership of array elements to the processors, distributes the charge of executing each instruction to the processor owning the variable modified by this instruction. Before a processor can execute some instruction, or group of instructions, it must have an up-to-date copy of all the variables which are owned by other processors and are needed for the execution. These update phases involve explicit processor communication in message-based run-time systems. The major weakness of this compilation model appears when the static data access analysis of the compiler produces poor results or fails. In this case, large amounts of the distributed arrays become potentially accessible during parallel loop execution, generate expensive communications during the update phases and can result in complex memory management at run-time.

It is possible to map the HPF distributed arrays on some form of shared memory. As long as the views of this shared space are consistent from all processors, the update phases can be removed and using simple synchronization barriers is sufficient to respect the data dependencies. On a demand-paging distributed shared memory, only the pages containing data accessed during a parallel loop execution need to be brought in a processor. The behavior of such a system is not directly dependent on the data access analysis of the compilers. However many authors have reported poor performances on distributed shared memories for various reasons:

- **false sharing:** independent data share the same page,
- **latency:** interprocessor communication latency dominates page-fault resolution,
- **sequential consistency:** expensive communication protocol through page invalidation before allowing modifications to shared pages,
- **weak locality in DSM organization:** mismatch between the processors in charge of managing a page and the processors accessing the page.

The independence of the iterations of a parallelized loop guarantees that the data modified during one iteration need not be propagated for the execution of the other iterations of the same loop. It is possible to relax the constraint of maintaining a strict sequential consistency during parallel loop execution. But, in this case, the

DSM must offer some means to manage the consistency of the shared pages in order to respect the data dependencies of the parallel programs.

Projects such as TreadMarks [1] or Munin [4] have shown that the use of a relaxed consistency software DSM along with some consistency control on synchronizations can avoid explicit and complex updates of the local memories. Using the *release consistency* model with a *lazy invalidate* protocol implemented in TreadMarks, the views of the pages from the processors become automatically consistent each time the processors synchronize. As long as the parallel codes are race-free, release consistency guarantees the same results as sequential consistency. With *entry consistency* [2], shared objects are associated with synchronization variables. This consistency management is more selective but requires some data access analysis in order to associate objects and synchronization variables.

Mome (**Mo**dify-**me**rge) is a software relaxed consistency, multiple writers distributed shared memory. Mome mainly targets the execution of programs from the High Performance Computing domain which exhibit loop-level parallelism. Using Mome, a processor must make an explicit consistency request on a shared memory sections each time its view of this section must integrate the modifications from other processors. For the compiler point of view, the strategy for Mome is close to the strategy for an explicit message passing run-time system: the compiler must evaluate which sections of the memory must be up-to-date before the execution of a parallel loop.

The next section of this paper outlines the use of the owner-compute rule for the distribution of the parallel computations on the processors of a distributed architecture. Section 3 presents the Mome DSM and its consistency model. Some preliminary execution results are presented in section 4.

2 Ownership Distribution and Owner-Compute Rule

The owner-compute rule consists of associating one processor, the *owning* processor, of the parallel architecture to each element of the distributed arrays and to restrict the ability to update (modify) this element to this sole processor. As long as this rule is applied, the owned elements are always up-to-date on a computation node of a distributed memory computer and need no communication or synchronization before being accessed by this processor. On the other hand, before a pro-

cessor is allowed to read some non-owned variables (only read accesses are possible as long as the owner-compute rule is applied) from its local memory, the current value of this variable must be updated from the owning processor. This rule can be applied for the execution of simple instructions on a parallel architecture. When the owner-compute rule is applied for the distribution of the iterations of a parallel loop on the processors, a processor executes all iterations which modify owned elements. All the communications necessary for the update of the non-owned read data are performed before the loop execution. The application of this simple scheme allows for efficient compilation of a large family of HPF programs. However, a strict application of this rule can be difficult, or impossible, in some cases:

- complex indexing on left hand side: complex distribution of the iteration space;
- complex indexing on right hand side: difficult to characterize which parts of the address space must be updated;
- multiple assignments inside the loop body: it is possible that all the elements updated by the same iteration are not owned by the same processor.

Many techniques have been developed to handle these complex cases, for instance, loop transformations or duplicated executions with masking of the updates of non-owned elements. Another solution consist of relaxing the strict application of the owner-compute rule. But in this case, the processors must be synchronized after the execution and must communicate their non-owned but updated elements before the compilation process can be applied to the remaining part of the program.

Owner-Compute Rule and Software Memory Consistency

The data update sequence inserted by the compiler before the loop executions can be considered as a form of explicit software memory consistency management. The efficiency of this update phase depends mainly on the quality of the data dependency analysis of the compiler. An inaccurate evaluation can lead to the transfer of large volumes of data and finally to poor performance.

Owner-Compute Rule and Distributed Shared Memory

No need for explicit consistency requests if the DSM is sequentially consistent. Only synchronization barriers need to be inserted before or after the parallel loops

in order to enforce the data dependencies. The DSM still gets benefits from the owner-compute rule applied by HPF compilers: this rule increases the data locality on the processors. However, as it has been shown by many authors, a sequentially consistent DSM suffers from false sharing and from the cost of page invalidation. False sharing appears when updates to some page by one processor conflict with read or write accesses to different locations of the same page on other processors. False sharing produces invalidations of the page on the processors and results in weak performance.

The use of a relaxed consistency DSM avoids the mutual invalidations in case of false sharing. The consistency management of a DSM can be implemented using a two-steps procedure. The first step removes all access privileges of the application to the memory section which must be made consistent. This first step is local to the requesting processor and generates no processor communication or data transfer. The second step is initiated when the program touches the memory section and generates a page-fault. This second step can result in processor communication and data transfer. This step is applied only on the pages containing data accessed by the processor. This two-step procedure limits inter-processor communications and data transfers to the sole pages containing useful data even in case of weak data dependence analysis at compile-time.

The code generation technique for such a relaxed consistency DSM is close to the code generation technique for a pure distributed memory architecture based on message passing: the main difference is that the explicit software management of the local memories through data communication is replaced by consistency requests on the memory sections which must be up-to-date. Moreover, in the case where the strict application of the owner-compute rule must be relaxed, the presence of a DSM allows a more simple construction of a consistent view of the data: at the opposite of the message passing version, there is no need to know which processor modified which data.

3 Mome DSM Consistency Model

Basically, Mome implements a simple relaxed consistency model for the DSM page management. At any time, the DSM considers a current version of each page. In the absence of explicit consistency requests from the processors, the current version

of the page is forwarded to any processor requesting a copy of the page, after a page fault. As long as no processor is granted write access, all processors having access to the page hold an identical copy. The processors requesting write access to the page can be allowed to simultaneously modify their local copy. The sole restriction is that, when two or more processors can modify a page, at least one unmodified copy of the page must be kept in the system. This original copy is necessary when the modifications are merged for the creation of the next version of the page. In general, the Mome DSM allows the first writer to modify its local page and requests the second writer to keep a copy of the original page.

3.1 Consistency Control

The Mome DSM allows the application to specify which sections of the shared memory must be made consistent on each processor, mainly after synchronization operations. The consistency control of Mome is based on a global distributed clock maintained by the DSM. Mome keeps track of the dates of some events: the date of the last synchronization barrier, the date of the last release of a distributed lock or the date of the first modification to the current version of each page. Each consistency request makes reference to the date of some event in the past. The HPF compilation strategy for Mome inserts a synchronization barrier followed by consistency requests before each parallel loop. These consistency requests use the last barrier date as a reference for consistency checking. During the first step of a consistency request handling, the page is protected from the application and the constraint date is stored in the page descriptor. This date is transmitted to the global page manager during the second step after a page fault. The page manager then compares this date to the current modification date associated to the page. The current version is valid if no modifications were allowed before the constraint date. In the other case, the manager initiates a merge of all pending modifications and generates the next version of the page. The current implementation of Mome generates the new version through a bit-wise exclusive or of the modified pages (and of the current version if the number of modified pages is even).

This two-steps procedure limits the updates to the pages which are really accessed by the application (a form of *lazy update*). Using a combination of consistency requests and prefetch requests (see section 4.4), the DSM interface allows to update the local memory before the application generates a page fault.

3.2 Shared Page Managers

Each memory page of Mome DSM is managed by a global page manager. The global managers are distributed on the processors. Mome provides the possibility to redistribute these page managers (see 4.7). The state of a page in a global manager is defined using four sets of processors: the set V of processors holding the current version, the set M of processors with modifications, the set S of processors which have asked for strong consistency and the set I of invalidated processors.

When a copy of the current version of the page needs to be forwarded to some processor, the sender is selected from V. It is possible for V to be empty only in the one-writer case and, in this case, M contains exactly one processor. M is non-empty if one or more processors have been allowed to modify the current version. If this set contains two or more processors, V cannot be empty: the current version of the page is necessary for merging the modifications. Set S records all processors having requested read or write access right to the page in strong consistency mode. These processors also appear in V or in M. As long as this set S is not empty, no other processor can be granted write access to the page. S can contain multiple readers or only one writer. In the case where S contains one or more readers, M is empty. If the global manager receives a write-request, all the processors from S are requested to invalidate their copy of the page and to forward a write acknowledgment to the future writer. The future writer is allowed to proceed only when it has received all the write acknowledgments. When all the processors use the strong consistency mode, the Mome DSM implements the classical multiple readers or one writer sequential consistency protocol. Set I records the processors which have modified the current version and forwarded their modifications. The current version is invalid for these processors because it does not contain their previous modifications. Any request from an invalidated processor generates a new version of the page integrating all modifications.

3.3 Controlling Mome from Data Access Analysis

As long as the owner-compute rule can be applied, the compilation strategy for Mome DSM consists of inserting a synchronization barrier followed by a consistency request on each accessed but not owned memory section before a parallel loop. This strategy is close to the message-passing strategy and can be implemented

with the same run-time interface. When the memory sections which are accessed during the computation can be determined with enough precision, it is possible to combine the consistency request with a non blocking prefetch request. The use of these requests initiate early page-fault resolution along with a possible merge of the pending modifications in the background and can reduce the latency of page-fault resolution. When it is not possible to characterize the accessed memory sections with enough precision, for instance in the presence of complex indexing of the arrays, the prefetch request can bring unused pages in the processor and should be avoided.

When the owner-compute rule cannot be applied, the parallel loop execution leads to the modification on non-owned elements in the shared memory. A simple synchronization barrier followed by a consistency request on the owned space, makes this space consistent. This scheme is similar to the explicit message passing scheme which explicitly updates the owned data.

At the difference of the message passing strategy, it may be necessary to introduce a synchronization barrier when write after read dependencies exist even when the parallel loop accesses only owned data.

4 Experiments

4.1 Simulation Codes

The effectiveness of using the Mome DSM for the execution of HPF-style programs was evaluated on two numerical codes: `tomcatv` and `mgs`.

During an external iteration of `tomcatv`, each data element of a two-dimensional grid is updated from the values of its neighbors in the previous iteration. Using HPF, all arrays of `tomcatv` are block distributed column-wise. The computation load of the processors is equitably distributed. Each processor reads one column updated by each of its two neighbors during each iteration.

`mgs` computes the modified Gramm-Schmidt algorithm. During external iteration `i` of `mgs`, column `i` is normalized and is then used for the orthonormalization of columns `i+1` to `n`. Different strategies can be used for the parallel execution of this algorithm. First, the array can be block distributed column-wise or cyclically distributed. Using block distribution, the computation load is unbalanced, especially at the end of the computation. During iteration `i`, only the processors owning

columns in the range `i+1...n` work. The use of a cyclic distribution balances the load on the processors but can result in a more expensive memory management. Cyclic distribution introduces false sharing on a DSM if the column length is not a multiple of the page size.

These two numerical codes were transformed manually. These transformations were limited to the computation of the parallel loop bounds and to the insertion of synchronization barriers and of consistency requests to the DSM before entering these loops. A version of the ADAPTOR [3] run-time library should be available in the future and allow to compare massage-based executions to DSM-based executions for the same source codes on the same hardwares. `tomcatv` was run on a 1023×1023 problem for the false sharing case and on a 1024×1024 when false sharing is avoided. For `mgs`, the results of Fig. 1 were obtained on 1023×1023 and 1024×1024 problem sizes and the results of Fig. 6 using 2047×2047 and 2048×2048 problem sizes.

4.2 Experimentation Parameters

The experimentation platform is a group of PCs connected through an SCI ring as well as a 100Mbits Ethernet network. Each node is a dual pentium II processor running SMP Linux 2.2.7. The following parameters were combined during the runs:

- communication layer: 100 Mbits Ethernet or SCI,
- false sharing: with false sharing or without,
- prefetch: with or without prefetching,
- consistency model: sequential or relaxed consistency model,
- consistency management strategy: on the whole DSM, on accessed elements or on non-owned accessed elements,
- localization of page managers: page cyclic, block column or column cyclic
- column distribution: block or cyclic for mgs

4.3 Communication Layer

During an external iteration of `tomcatv`, each processor crosses two synchronization barriers, receives one column from each of its two neighbors and computes its local block. When the number of processors increases, only the cost of the block